Similarity-driven Schema Transformation for Test Data Generation

Fabian Panse
University of Hamburg, Germany
fabian.panse@uni-hamburg.de

Johannes Schildgen
OTH Regensburg, Germany
johannes.schildgen@oth-regensburg.de

Meike Klettke
University of Rostock, Germany
meike.klettke@uni-rostock.de

Wolfram Wingerath
University of Oldenburg, Germany
wolfram.wingerath@uni-oldenburg.de

ABSTRACT

A flexible and versed generation of test data is an important aspect in benchmarking algorithms for data integration. This includes the generation of heterogeneous schemas, each representing another data source of the integration benchmark. In this paper, we present our ongoing research on a novel approach for similarity-driven generation of schemas, which takes as input an arbitrary dataset, extracts its schema, and derives a set of output schemas from it. In contrast to previous solutions, we do not focus on structural transformations of relational or XML schemas, but extend the scope to contextual transformations and NoSQL data models, where the required schema information is often only implicitly defined within the data and must first be extracted. In addition, we utilize a novel method that generates multiple schemas based on user-defined heterogeneity constraints making the generation process configurable even for non-experts.

1 INTRODUCTION

The number of available data sources and the need to integrate them is growing rapidly in many public, academic, and industrial sectors [15]. At the same time, the increasing diversity of the database landscape makes an accurate integration of these sources considerably more difficult. All these factors have made data integration [20, 21], and all its sub-steps, such as schema matching [4, 5], duplicate detection [17, 48, 50], and record fusion [8, 53], intensively studied research areas for decades, which still receive a lot of attention today (e.g., [11, 13, 24, 25, 34, 38, 44, 46]).

The development of novel algorithms for data integration requires a systematic and thorough evaluation of them [49]. This in turn requires test datasets that contain a ground truth. Real-world datasets with ground truths are hard to find, as this truth has to be determined very laboriously and at high costs. Therefore the use of test data generators is recommended, which allow a fast generation of different benchmarking scenarios and allow the users to create datasets according to their own needs.

An important step in generating test data for data integration is to create heterogeneous data schemas, each representing a different data source of the integration benchmark, and mappings between them. Typically, this is accomplished by transforming an input schema provided by the user. Current generators of schema-related data integration benchmarks, such as iBench [3], STBenchmark [2], or MatchBench [26], however, focus on structural transformations of relational or XML schemas explicitly defined in the given data, although such schema specifications are rarely complete or, in the case of many NoSQL datasets, missing altogether. Moreover, these tools are limited to the generation of schema pairs, despite the fact that real-world integration tasks often involve more than two data sources. Finally, their configuration requires detailed knowledge on schema-transformation operators and thus are difficult to use for non-experts.

In this paper, we propose a novel approach to generate data schemas (and mappings between them) for data integration benchmarks that in contrast to existing solutions

- also supports NoSQL data models, such as JSON or property graphs,
- profiles the input data to enrich explicit and extract implicit schema information,
- supports contextual schema transformations, such as changing a column’s format or unit of measurement, and
- utilizes a novel concept of similarity-based transformation trees to build an arbitrary number of output schemas that satisfy user-defined constraints on their heterogeneity.

The basic idea of our approach is illustrated in Figure 1. First, the user submits an arbitrary dataset (e.g., relational, JSON, or graph-based) as input along with its explicit schema (if available) and a configuration that specifies the desired heterogeneity of the output schemas to be generated. Second, the submitted dataset is profiled to identify, extract, and add missing schema information. Third, to simplify the subsequent generation of output schemas, the profiling results are used to decompose the input dataset and schema as much as possible. Fourth, the desired number of output schemas is generated by transforming the prepared input schema. To meet the user’s specifications on the output schemas’ heterogeneity as well as possible, we utilize a novel concept using similarity-based transformation trees. Finally, for each pair of schemas, two schema mappings as well as two transformation programs are generated, which will allow us later on to rewrite queries and transform data from one schema into the other. The
final output of our generation approach contains (i) the prepared input dataset and schema, (ii) \( n \) output schemas, and (iii) \( n(n+1) \) schema mappings and transformation programs between the individual schemas (input and output).

We plan to embed this schema generation approach into our DaPo generator [29], where we use the generated schemas to create benchmarks for duplicate detection and record fusion that consists of multiple data sources. However, the generated schemas, mappings, and programs can also be used to create benchmarks for other data integration tasks, such as schema matching/mapping [9, 34, 46], query rewriting [27], or data exchange [10].

The rest of the paper is structured as follows: First we elaborate on related work and open challenges in Section 2. Then we describe how we address schema profiling, preparation, and transformation in our research project in Sections 3 and 4. Thereafter, we discuss the calculation of schematic heterogeneities in Section 5 and present our overall generation approach in Section 6. We conclude this paper in Section 7.

2 RELATED WORK & OPEN CHALLENGES

Static benchmarks and experiment suites, such as Alaska [18], XBenchMatch [22], T2D [55], or Valentine [37], provide valuable test scenarios for different schema matching use cases. However, they are limited to particular domains (e.g., Alaska includes data on electronic devices) and can only be customized by removing individual schema components (e.g., sources, tables, or columns) from the test input. In addition, we cannot use them for DaPo, because it is supposed to work on arbitrary databases.

The schema generators STBenchmark [2], MatchBench [26], and iBench [3] all support the renaming of labels. While iBench and MatchBench are limited to relational schemas, STBenchmark also provides operators for transforming nested XML schemas. The transformation of constraints is partially covered by STBenchmark (referential) and iBench (unique, referential, and functional dependencies). All three generate benchmarking scenarios consisting of one source and one target schema. Thus, it is difficult to achieve a predefined degree of heterogeneity between multiple output schemas. EMBench++ [32] is a tool for generating entity matching benchmarks, which is able to add and remove individual columns from the benchmark’s schema, but does not support more complex schema transformations.

Besides benchmarking, there is a lot of work on schema and data transformation in many other research areas. Examples are data cleaning [54], schema evolution [19, 28, 36, 61], multi-database languages, such as SchemaSQL [39], or polyglot data management [16, 56]. Our goal is to reuse the results (e.g., transformation operators for NoSQL data models) obtained in these works whenever possible and to extend them if necessary. Despite the large amount of existing research on schema transformation, the following challenges remain for our project:

- Extraction of missing (implicit) schema information of a given dataset. This may be the whole schema if the data is managed by a schemaless NoSQL data store.
- Identification of appropriate operators for schema transformation and dependencies between them.
- Identification and collection of knowledge (e.g., ontologies) required for some schema transformation operators.
- Developing methods to measure the heterogeneity between two schemas.
- Developing an algorithm for generating multiple schemas while considering user-defined heterogeneity constraints.

3 SCHEMA PROFILING & PREPARATION

Every (semi-)structured dataset follows a schema, which is either explicitly managed by the underlying database system, implicitly defined by the data-processing applications, or a mix of both (e.g., an SQL database manages the dataset’s structure, but the semantics of some columns is only known to the applications). Due to evolving applications, in the latter two cases, different records of the same dataset may also conform to different schema versions [38].

3.1 Data Schema & Categories

In the literature, data schemas are often limited to structural descriptions. In this paper, we take a broader view of the term and consider the schema as the conglomerate of all information describing the actual data. We group this information into four categories: (1) structural, (2) linguistic, (3) constraint-based, and (4) contextual. Linguistic schema information refers to (the semantics of) labels, such as the names of relational tables and columns, XML tags, or field names in field-value pairs (e.g., ‘(name,’Jan’)). Constraint-based information refers to integrity constraints ranging from keys to application-specific conditions. Contextual information encompasses all remaining information necessary to fully interpret individual data objects (e.g., tables, columns, or values). For example, the context of a column includes its (i) format (e.g., ‘yyyy-mm-dd’ vs. ‘dd.mm.yy’), (ii) level of abstraction (e.g., district vs. city), (iii) unit of measurement (e.g., ‘cm’ vs. ‘inch’), and (iv) encoding (e.g., [yes,no] vs. [1,0]). The context of a table includes its scope (e.g., ‘book’ vs. ‘novel’).

3.2 Data & Schema Profiling

Many datasets do not contain an explicit description of their complete schema. However, the more detailed schema information we have, the greater the choice of transformation operators we can apply to it. Thus, it is important to derive a schema from the input data that is as accurate, complete, and detailed as possible. The profiling of data is currently a hot topic in the database community [1], and there is already a lot of (ongoing) research on identifying and extracting (i) schema information in/from CSV files [33], JSON documents [35], and graph databases [40], (ii) integrity constraints, such as unique [7], denial constraints [45, 52], inclusion dependencies [59], or functional dependencies [6, 14, 51, 57], and (iii) semantic domains [31, 62], that we reuse for our purpose. However, the identification of some contextual information, such as the scope of a table or the unit of measurement of a column, has not yet received much attention and needs further research. The same applies to identifying the semantic closeness of columns to determine which of them are likely to merge.

3.3 Data & Schema Preparation

After we have profiled the input data, the obtained schema information is used to further decompose the input dataset and schema so that their information is represented in as much detail as possible. This decomposition has the goal to simplify subsequent transformations. For example, it is easier to merge two attributes than to split one. Therefore, we transform the input dataset into a structured data model, normalize its schema, and split its attributes into several subattributes if a clear separation between the corresponding values is possible. Moreover, if its records conform to different schema versions, they are all initially migrated to the same version (e.g., the latest one) [36].
same may apply if we increase the level of abstraction of a column (drill-up). Thus, we have dependencies between the four aforementioned categories. Typically, a structural operator implies a linguistic or contextual operator, and a contextual operator implies a linguistic one, but not vice versa. In addition, changing a context may require to change an integrity constraint. For example, when converting the unit of measurement of a column from ‘feet’ to ‘cm’, we may need to adapt a constraint that restricts the maximum size value. Finally, linguistic transformations also often require a refactoring of constraints. This leads us to the following (approximate) dependency order:

\[
\text{structural} \rightarrow \text{contextual} \rightarrow \text{linguistic} \rightarrow \text{constraint}
\]

4.2 Required Knowledge
Several transformation operators require additional information, which we store in a knowledge base (see Figure 1). Most structural transformations only require knowledge about the data model with which the given schema is defined. It becomes more complex if the schema has to be transformed from one model (e.g., relational) into another (e.g., JSON). In this case, we need transformation rules, either directly between both models (e.g., [41, 56]) or indirectly via a third model, which can be a generic one such as U-schema [12]. In addition to these mappings, we need dictionaries and ontologies (e.g., from DBpedia [43]) to enable linguistic and contextual transformations addressing semantic relations, such as synonyms or hyperonyms. For changing units of measurement, we need conversion rules, which in turn may be time-variant (e.g., the daily changing exchange rate between two currencies). Finally, changing the format or encoding of a column requires alternative (and common) representations and terminologies of the corresponding domain, which we collect from other datasets, such as the Dresden Web Tables Corpus [23] or GitTables [30].

5 HETEROGENEITY CALCULATION
Systematic benchmarking requires that the user is able to generate test datasets with varying degrees of heterogeneity. To assist inexperienced users who are unable to map such a degree to a corresponding sequence of transformation operators, we need measures to quantify the heterogeneity between the generated output schemas. Since heterogeneity can be seen as the conceptual opposite of similarity, we can use common similarity measures for this purpose. Such measures can greatly differ from one schema category to another. For this reason, we separate our measurement into four parts accordingly and model the heterogeneity of two schemas by a quadruple \( h \in [0,1]^4 \) where each of the tuple’s values represents the normalized heterogeneity with respect to one of the four schema categories (see Section 3.1).

Subsequent calculations with those quadruples follow the rules of component-wise addition and scalar multiplication, i.e., let \( v, w \in \mathbb{R}^4 \) and \( \lambda \in \mathbb{R} \), it holds:

\[
\pi_k(v + w) = \pi_k(v) + \pi_k(w)
\]

\[
\pi_k(\lambda \cdot v) = \lambda \cdot \pi_k(v)
\]

where \( \pi_k(v) \) gives the \( k \)th entry of tuple \( v \). Moreover, it holds:

\[
\pi_k(op(v, w)) = op(\pi_k(v), \pi_k(w))
\]

for \( op \in \{\min, \max\} \).

The meaning of structural similarity between two schemas strongly depends on the available structures and thus can greatly differ between the individual data models. Existing measures
Moreover let $\sigma_i \in \mathbb{R}^4$ be the total sum of heterogeneity we still need at the start of run $i$ to meet the user’s specification. Initially, it equals $\sigma_1 = \rho_1 \cdot h_{c_{\text{avg}}}$ and decreases by $h_i = \sum_{j=1}^{i-1} h(S_i, S_j)$ after we have generated schema $S_i$ in the $i$th run. i.e., $\sigma_{i+1} = \sigma_i - h_i$. Based on these numbers, the two thresholds are calculated as:

$$h_{\text{min}}^k = \max(h_{\text{min}}^k (\sigma_i - \rho_{i+1} \cdot h_{c_{\text{avg}}}) / (i - 1))$$  

(7)

$$h_{\text{max}}^k = \min(h_{\text{max}}^k (\sigma_i - \rho_{i+1} \cdot h_{c_{\text{min}}}) / (i - 1))$$  

(8)

6.2 Generation of Each Output Schema

Due to the dependency order described in Section 4.1, the generation of each output schema $S_i$ is executed in four steps:

(1) structural transformations
(2) contextual transformations
(3) linguistic transformations
(4) constraint-based transformations

Between every two steps, dependent transformations of the following categories are identified and executed.

In each step $k \in \{1, \ldots, 4\}$, we span a so-called transformation tree (see Figure 3). The root node $n_0$ represents the schema resulting from the previous step (or the prepared input schema if $k = 1$). This node is expanded by applying a predefined number of transformations. The resulting schemas form the child nodes of $n_0$. Then, for each of these schemas $S$, the heterogeneity to all already generated output schemas is measured regarding the schema category of the respective step (i.e., structural if $k = 1$). The result is the heterogeneity bag $H_k(S) = \{\pi_k(h(S, S_j)) \mid j < i\}$. The node of schema $S$ is called valid if

$$\forall h \in H_k(S): h \in [\pi_k(h_{\text{min}}^k), \pi_k(h_{\text{max}}^k)]$$  

(9)

In addition, a valid node is called a target node if

$$\text{avg}(h(S_j)) \in [\pi_k(h_{\text{min}}^j), \pi_k(h_{\text{max}}^j)]$$  

(10)

Next, a leaf node of the current tree is expanded. If there is already a target node by then, the node to be expanded is selected randomly among all leaf nodes. If this is not the case, the distance to the range $[\pi_k(h_{\text{min}}^j), \pi_k(h_{\text{max}}^j)]$ is calculated for all leaf nodes, and then the leaf node with the smallest distance is expanded.

The construction of the tree ends after a predefined number of nodes have been expanded. If there are target nodes by then, one of them is chosen randomly as the output schema $S_i$. Otherwise, the schema of the node with the smallest distance is returned as output where valid nodes are preferred to non-valid ones.

7 CONCLUSION & ONGOING WORK

In this paper, we presented a novel approach for generating a set of heterogeneous data schemas as needed in data integration benchmarks. In contrast to existing solutions, it is (i) not limited to relational or XML schemas, (ii) able to deal with implicit schema information, (iii) also considering contextual schema transformations, (iv) able to generate scenarios with more than two schemas, and (v) similarity-driven, i.e., the generation is based on user-defined similarity scores, which ease its configuration.

The research presented in this paper is ongoing work which is being implemented in the context of the DaPo project [29]. In this project, we use the resulting schemas as input to a data pollution process to generate realistic benchmarks for duplicate detection and record fusion consisting of multiple heterogeneous data sources. The next steps of the project include the development of (i) similarity measures for the different schema categories, and (ii) a filter that selects suitable transformation operators depending on the respective node of the transformation tree.


